DETR with Modulated Object Queries For Object Detection

Sudeep Narala
Department of Electrical Engineering
Stanford University
sudeepn@stanford.edu

June 3, 2022

Abstract

Object detection has been one of the fundamental problems in computer vision. It has numerous applications including automatic analysis of CCTV footage, automated retail checkout and vehicle detection just to name a few. Detection with Transformers (DETR) has been a groundbreaking contribution in this space. Though DETR achieves similar results to R-CNN, it suffers from the problem of often struggling to detect smaller objects (and compensates by better detection of larger objects). In this paper, we discuss approaches to help DETR learn from itself and use its strengths in detecting larger objects to help it in detecting other objects by using 2 passes through the decoder. In order to aid in this self-learning process, we explore GNNs and other transformation techniques.

1 Introduction

In the problem of object detection, the input to the model is an image and the output is a set of bounding boxes on the image with class designations for each box.

The first important solution in this space was that of R-CNN and the subsequent Faster R-CNN [7]. However, more recently DETR (Detection with Transformers) has been used to achieve similar results to R-CNN while removing the need for hand-designed components such as non-maximal suppression and anchor generation, as used in R-CNN and its variants, and replaces them with a set matching problem based on the transformer output instead. Moreover, more recent state of the art work has built on top of DETR.

While the transformation architecture in DETR has the added advantage that it takes the whole image into account for detections, it still suffers from problems. As mentioned in the original paper, DETR suffers from a fundamental problem of not being able to perform well on small objects, which the authors hope can be remedied in future work. In this paper, we will be introducing a few methods to try to enable DETR to be better with small object detection using larger objects. In a sense, the model will be taught to "learn from itself" to make better detections.

2 Related Work

2.1 Detection with Transformers

This paper builds on the DETR model [1]. DETR consists of a CNN backbone and an encoder-decoder transformer [8]. The CNN backbone we used is Resnet50. DETR takes the features outputted by the backbone and passes them through a 1x1 convolution to shrink the number of channels. Next, it flattens the spatial dimension and passes these through an encoder module. More formally, if the dimension of the output from the backbone is (C, H, W), DETR first converts this to (d, H, W) and then flattens along the spatial dimension to create (d, HW). Now, this is passed into the encoder where HW serves as the

encoder sequence length. Next, the decoder takes a fixed number of "object queries". This number is set to a number much larger than the typical number of ground truth detections in an image. The detector takes the output of the encoder as input and the object queries as input. This object query can be viewed as a learned positional embedding (and the input sequence to the first decoder layer is simply 0s).

2.2 Two-pass Decoding

Two-pass decoding has been studied in the field of NLP. In the paper, Deliberation Networks: Sequence Generation Beyond One-Pass Decoding [11], the authors show how the process of "deliberation" is very important for polishing a prediction by taking the whole context of the target into account. Though we borrow the basic idea, we chose a very different implementation. In [11], the first pass and second pass decoder are distinct. The second pass decoder uses the context (i.e. hidden states) generated from the first pass decoder in order to generate its final prediction at the final time step. However, in our case, the decoders use shared weights and the first decoder simply helps modulate the input queries to the second decoder.

2.3 Importance of Object Query Construction

This paper also borrows ideas from Conditional DETR [5], Anchor DETR [10] and [2]. These papers showed the importance of carefully chosen (or learned) object queries, instead of simply a learned a positional embedding, to boost convergence speed. Conditional DETR [5] did this by localizing each object query by changing the attention mechanism from the original transformer. Anchor DETR [10] instead explicitly sought to make each object query serve as an anchor point. This also localized the predictions from each object query and helping with training convergence. Anchor DETR makes use of an additional learned embedding, in addition to the regular object query embedding, in order to predict multiple objects at each anchor point. For example, if one wanted to allow 3 predictions per anchor point, Anchor DETR learned 3 additional positional embeddings that were repeated and added to the regular learned object query embedding at each spot (thus effectively increasing the number of object queries by 3 times). We will be using a similar concept in "modifying object queries without spatial awareness". SMCA (Spatially Modulated Co-Attention) [2] achieved 10x faster convergence by using projected encoder outputs and a spatial prior based on self attention among object queries to create a new object query. This again illustrates how important a carefully constructed object query is.

2.4 Techniques for Object Query Construction

We also used some methodologies similar to Spatial-aware Graph Relation Network for Large-scale Object Detection [12]. This paper used GCNs [3] with learned edge weights and spatial awareness with gaussian kernels (dependent on distance and angles between anchors) to make better feature proposals. We will be making use of GCNs as well in one of our approaches.

3 Methods

The key problem we were trying to address is getting better detections for smaller objects. The approach was to try to use the model to "learn from itself". This can also be viewed as auto-regression to use the model's outputs to bias it to make more right predictions.

Let's introduce some values. First off, DETR makes use of a value N, which is the number of object queries in the decoder. The Each object query is some learned positional embedding denoted by:

$$O_q = Embedding(N, hidden\_dim)$$

where $O_q[i]$ gives you the i-th object query. The decoder makes use of the encoder features in the cross-attention module and uses the object query as a positional encoding that is added to the target sequence. As previously mentioned, the target sequence starts

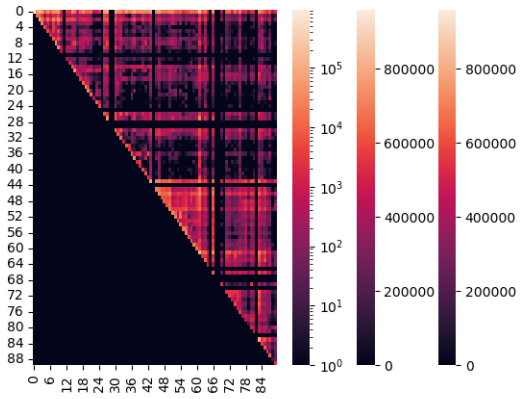

Figure 1: Co-occurrence Heatmap Between Coco Classes

out as all 0s in the 1st decoder layer. To the read unfamiliar with the transformer architecture, we refer [8].

Finally, one aspect of DETR that is important for us is that it uses a class predictor and bounding box regressor. The class predictor (*class_pred*) is simply a linear layer while the bounding box regressor is a 2 layer MLP (*bbox_reg*).

We tried 2 methods in order to achieve this end goal of self-learning by extracting more global information inferred by the model. We will be making 2 passes through the decoder in both approaches. One with standard positional embeddings and one with modulated positional embeddings.
When looking at both these methods, it is important to note that DETR does an excellent job with larger objects, but struggles with smaller objects. How can we make DETRs confident, and correct, knowledge of larger objects help us predict smaller objects that may have been missed or incorrectly predicted? The basic nature of both the approaches is depicted in 2. We are going to be looking at 2 approaches we designed for the Modulation Network.

3.1 Modified Object Queries Through GCNs

Our goal here is to use the predictions made by the model in the first pass through the decoder to refine the object queries. The intuition we will need here is that we need to take into account which object query led to the prediction, what the class prediction was and what the bounding box predicted was. This will help give the model spatial awareness, and help information propagate between the object queries to lead to better starting positional encodings to the decoder. As was shown in [5] and [10], this alone can be very crucial.

In this approach, we model each object query as a node. We first assign distinct node features for each object query:

$$G_n = Embedding(N, gcn\_input\_dim)$$

Now that we have the node embeddings, we need edge weights to form the graph. One thing to note here is that each image (data sample) will have a distinct learned graph. We will be learning edge weights, with a similar idea to the work from [12] with Faster R-CNN like methods.
As done in [12] and [9], we use the outputted probabilities from class predictor as features for the node. We also use the predicted bounding box coordinates. We need to introduce a new set of embeddings for each object query to serve as a feature as well:

$$G_f = Embedding(N, gcn\_feature\_dim)$$

Note that this is a separate embedding than the node features.

Suppose the output from the decoder on the first pass is d_o. Now, the overall feature for each node i becomes:

$$F[i] = G_f[i] \mid class\_pred(d_o) \mid bbox\_reg(d_o)$$

Now, we need to use this feature to predict edge weights to every other node in the graph (i.e. other object query nodes). We opted to use a 2-layer MLP to do this. Similar to [12], we only pick the top k (a hyperparameter) edge weights and form our graph.

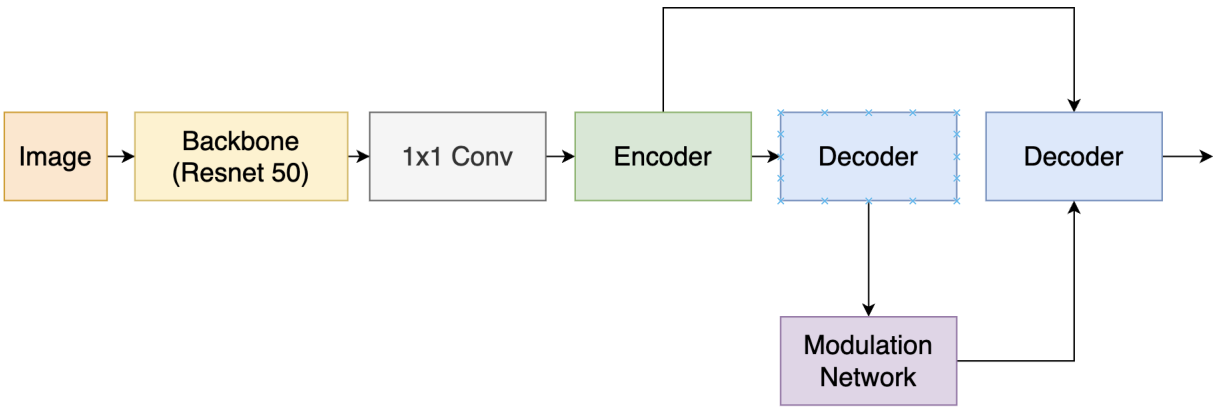

Figure 2: Block Diagram Of Approach

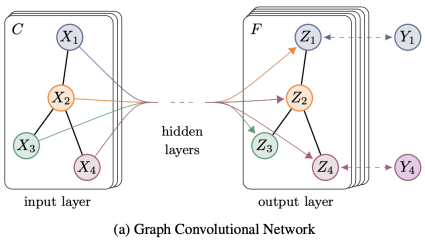

Figure 3: Depiction of a GCN from [3]

Finally, we pass these edge weights and node features through a 2-layer GCN (i.e. the computational graph relies on the node's neighbors and the neighbors of the node's neighbors) depicted in 3. Further details of GCNs as left out of this paper and can be found in [3]. The GCN is expected to learn the intricacies formed by the graph and relay that into modulating the object queries.

3.2 Modified Object Queries Without Spatial Awareness

Intuitively, one observation we made was to consider that object placement didn't matter as much as object co-occurrence. This relationship might also be simpler to teach the model explicitly, so we wanted to try it. We like to view this problem as trying to modify the apriori assumptions the decoder can make about the semantics of the image in the 2nd pass using the 1st pass.

First, we have to create a set of class embeddings in order for it to learn (potentially complex) relationships between classes:

$$c_e = Embedding(C, class\_embed\_dim)$$

where C is the number of classes in the dataset + 1 (for the background class).

We first take the maximum probabilities and classes predicted by the first pass through the decoder. Now, we take a linear combination of the class embeddings weighted by the probabilities predicted by the decoder:

$$F = max(softmax(class\_pred(d_o)))$$

$$\cdot\, c_e[argmax(class\_pred(d_o))]$$

Finally, we pass this through a 2 layer MLP in order to get the offset (i.e. modulation) of all the object queries from their base, learned positional encodings. There are 2 caveats to this. First, we ignore all cases where the predicted class is the background class because this would just dominate F and would lead to room for signal from real predictions. Second, we only modify the base, learned positional encodings if the decoder wasn't confident in its prediction for that

object query before OR if the predicted class was the background class.

Now, the new input to the first layer of the decoder becomes:

$$O_q + MLP(F)$$

where $MLP(F)$ is unsqueezed and repeated (broadcasted) across all the base object queries. This is an idea similar to [10], where they learned an additional set of embeddings to predict multiple objects at the same anchor point and added them uniformly across base object query embeddings.

This methodology gives us the positional encodings (which will be combined with the original object queries) for the second pass through the detector, with hope that we have taken into account the semantics of the image. This allows the model to only reconsider detections that were predicted as the background class or detections it wasn't confident in during the first decoder pass.

4 Dataset and Features

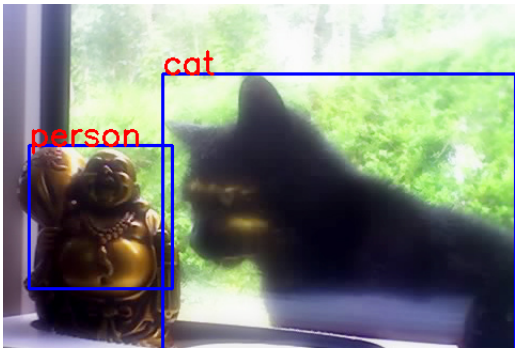

Figure 4: Created using this colab that we wrote.

We used the Microsoft COCO (Common Objects in Context data) [4] dataset. The dataset contains annotations for both panoptic segmentation and 2-D object detection, both of which can be performed by DETR. However, we will only be using the 2-D object detection in this paper. The 2-D object detection annotations consist of a set of bounding box coordinates and corresponding class labels for each

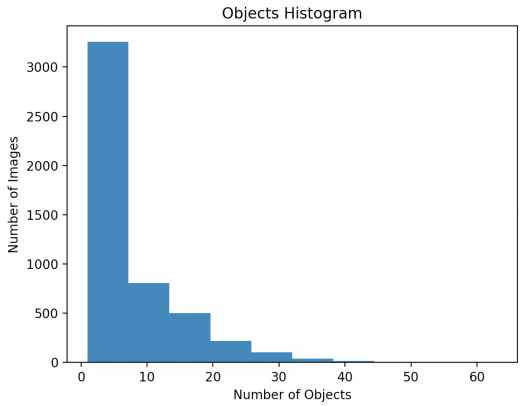

Figure 5: Histogram of the Coco Dataset showing how many objects are in each image

image. An example image with the bounding boxes of objects drawn over, with the corresponding class labels, is depicted in 4.

Let's do some quick analysis on the dataset. 5 shows object frequency in the dataset. This informs the choice in DETR to use a maximum of 100 object queries in the decoder, which corresponds to a maximum of 100 detections. As we can see, no image in the Coco dataset has this number of objects in it. As mentioned before, 1 also shows interesting relations in the dataset showing trends in co-occurrence among objects within an image.

5 Experiments

We forked the DETR repo to perform experiments. All code was written in pytorch [6].

Unfortunately, we struggled with the experiments for this project. One epoch through DETR was taking 6 hours when trained end-to-end with the Resnet50 backbone. In order to train this model, we had to make several concessions which definitely impacted the overall performance. We also were unable to try new experiments because of the sheer amount of time it took to train. Note that DETR took 500 epochs to reach training convergence.

Since most of our experimentation used the pre-trained DETR as a starting point, we hypothesize

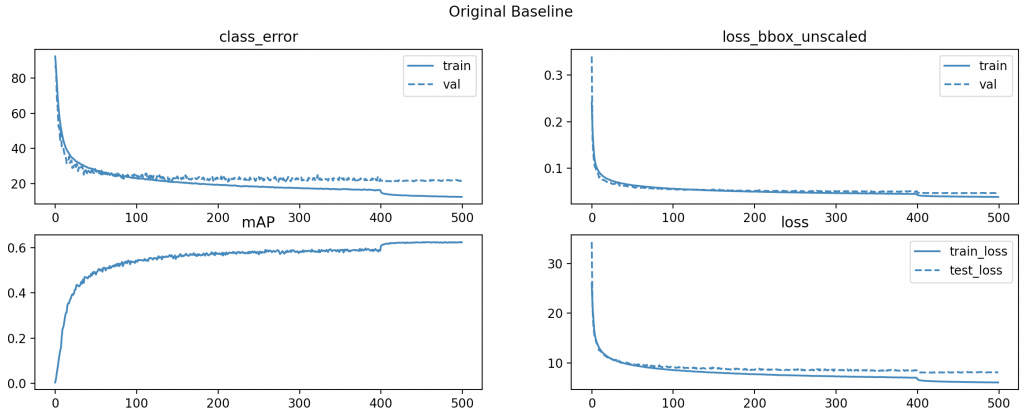

Figure 6: Baseline Statistics from Original DETR. Dashed lines depict val results. mAP is only calculated on the val set.

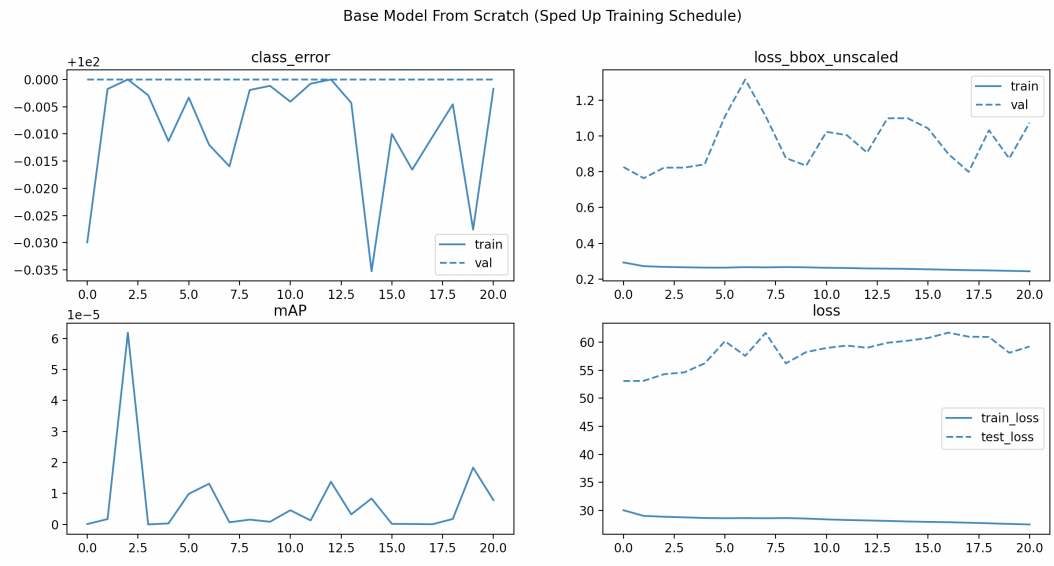

Figure 7: Training from Scratch Statistics. Dashed lines depict val results. mAP is only calculated on the val set.

that even if our methods were sound in theory, they wouldn't yield good results. This is because after 500 training epochs, DETR's is highly intertwined on having the specific learned object queries it was trained for. Modulating them did not help improve performance because it would "undo" a lot of the training that led it to perform so well in the first place. We discuss this further in the conclusion, and how we hypothesize that an alternating training strategy would have probably suited this style of solving the problem better. However, this would require training from scratch, an approach which we

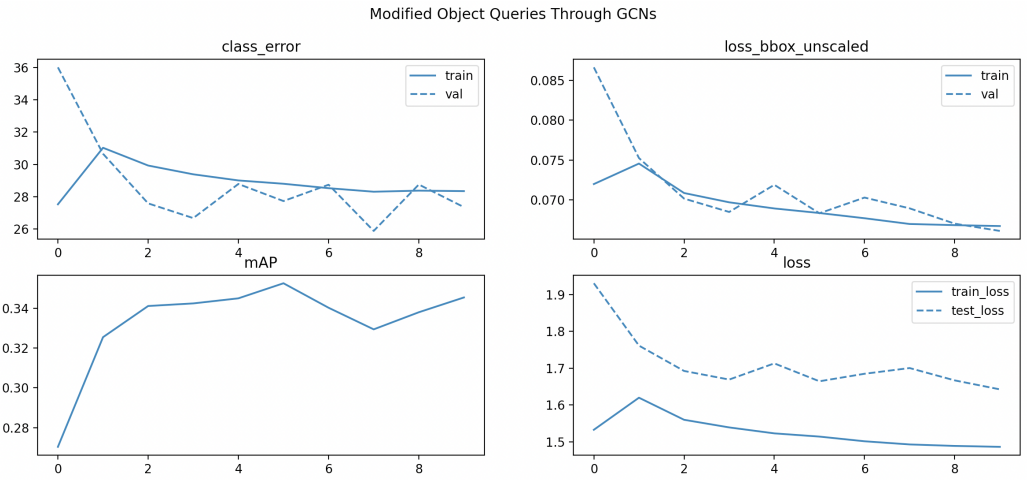

Figure 8: GNN Training Statistics. Dashed lines depict val results. mAP is only calculated on the val set. Note that the loss scale is way lower than other training statistics because we didn't use augmented loss (which they do in [1]) here to improve training speed.

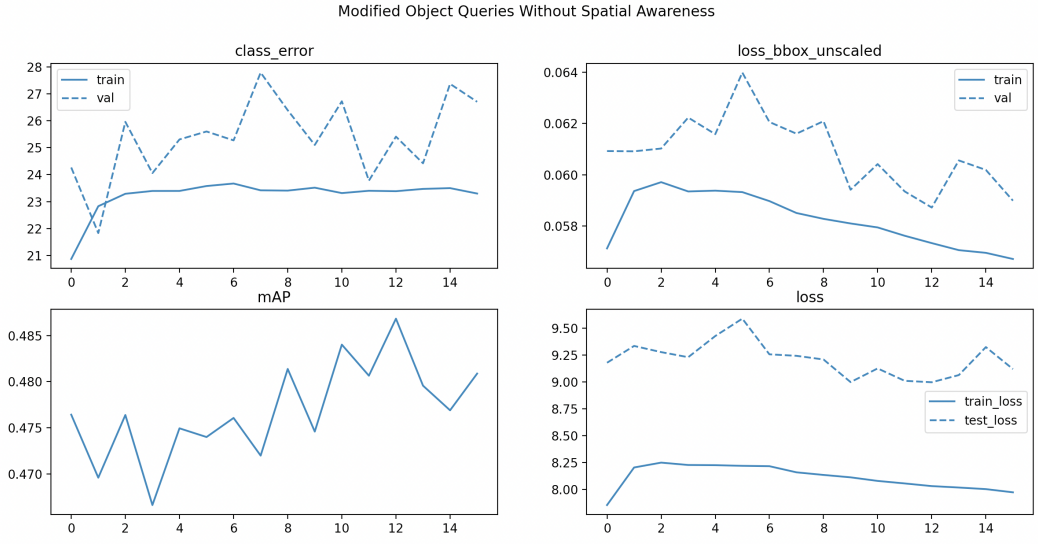

Figure 9: Modulation of object queries without considering spatial nature. Dashed lines depict val results. mAP is only calculated on the val set.

will see was not viable.

5.1 Training From Scratch

We first attempted to train DETR from scratch for a few epochs on part of the dataset. Since we wanted to

make this as quickly as possible, we couldn't perform end-to-end training through the backbone. Thus, we used the backbone outputs as features into DETR without backbone. To make this efficient so we weren't computing the forward pass for each epoch repeatedly, we cached the results of a pass through the backbone on disk. We also didn't want to train with all the data, so we used only 25% of it selected randomly. For 25% of the images (about 29500 images) in the training set, the size of this cache was about 50GB.

Using these cached features, we attempted to train DETR from scratch. We didn't expect to see good results, but we did expect a very coarse baseline with which to work with. This did not go as planned. As shown in 7, we were unable to get anything out of the model in a reasonable amount of time. In fact, the training loss was increasing Thus, we were forced to abandoned this idea.

If we were able to get this to work, we would've tried the alternating training strategy that we think would have proven to be more successful.

5.2 Modified Object Queries Through GCNs

For this experiment, we started from the pretrained weights posted on the DETR github.

This method proved slightly more promising. However, we were not able to reach the baseline in a reasonable amount of time. We did, however, see an uptrend during the training process as the features were being properly learned. As shown in 8, we do see a slight downtrend during training. The mAP seemed like it was on an uptrend. However, it is still not even close to the baseline mAP of 0.624 from 6. Due to time considerations, we were forced to terminate the run.

We hypothesize that an alternated training schedule would prove to be far more ideal. However, due to the lack of success observed training from scratch, we were unable to try this schedule.

5.3 Modified Object Queries Without Spatial Awareness

In this method, we trained in a similar fashion. We, again, started with pretrained DETR weights.

We see the results of the training process in 9. We note that the training loss seemed to be following a healthy pattern after 15 training epochs. The mAP also seemed to be increasing steadily, but was still nowhere near the baseline. However, this alone took about 2.5 days of continuous training (including the val step after each epoch). We were forced to stop this run because we ran out of time. However, this result seemed the most promising.

6 Conclusion

We should have noted the computational constraints in working with DETR earlier in the project given our resources. We were unable to train end-to-end. Even with this restricted training, we were only able to train for 10s of epochs when we would need much more to perform any experimentation.

As discussed earlier, our methods were detrimental to performance since they were making the model unlearn the object queries which were so intertwined with the high model performance. However, papers [5] and [10] showed the importance of more carefully chosen object queries. We think that if we trained the model in an alternating way, i.e. we alternate between a single pass through the decoder and two passes through the decoder, we could get better results because the "self-learning" component and the regular model would learn to work together while the first pass decoder still learns to gather good results by itself.

Another mistake we made in the paper was not using a pairwise scoring function (like a dot product) for edge weight modeling. Using just an MLP on each object query feature was probably not the right approach. Given more time, we would definitely try this with an alternating training schedule.

7 Acknowledgements

We forked off DETR in order to implement our changes. The original code can be found here.

References

[1] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European conference on computer vision*, pages 213–229. Springer, 2020. 1, 7

[2] Peng Gao, Minghang Zheng, Xiaogang Wang, Jifeng Dai, and Hongsheng Li. Fast convergence of DETR with spatially modulated co-attention. *CoRR*, abs/2101.07448, 2021. 2

[3] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 2, 4

[4] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 5

[5] Depu Meng, Xiaokang Chen, Zejia Fan, Gang Zeng, Houqiang Li, Yuhui Yuan, Lei Sun, and Jingdong Wang. Conditional detr for fast training convergence. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3651–3660, 2021. 2, 3, 8

[6] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 5

[7] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015. 1

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 1, 3

[9] Xiaolong Wang, Yufei Ye, and Abhinav Gupta. Zero-shot recognition via semantic embeddings and knowledge graphs. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6857–6866, 2018. 3

[10] Yingming Wang, Xiangyu Zhang, Tong Yang, and Jian Sun. Anchor detr: Query design for transformer-based detector. *arXiv preprint arXiv:2109.07107*, 2021. 2, 3, 5, 8

[11] Yingce Xia, Fei Tian, Lijun Wu, Jianxin Lin, Tao Qin, Nenghai Yu, and Tie-Yan Liu. Deliberation networks: Sequence generation beyond one-pass decoding. *Advances in neural information processing systems*, 30, 2017. 2

[12] Hang Xu, Chenhan Jiang, Xiaodan Liang, and Zhenguo Li. Spatial-aware graph relation network for large-scale object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9298–9307, 2019. 2, 3